GAZDASÁGINFORMATIKUS FELSŐ OKTATÁSI SZAKKÉPZÉSI SZAK ZÁRÓVIZSGA TÉTELEK

Érvényes 2018/2019. tanévtől

„A" TÉTELSOR (MATEMATIKAI ÉS INFORMATIKAI INTÉZET TANTÁRGYAI)

1. Bevezetés az informatikába: Az információ fogalma, mérése, kódolása. Kódrendszerek. A gépi információ (adat, utasítás) ábrázolása, számábrázolás, számítógép felépítése, alacsony és magas szintű programozási nyelvek (generációk).

Az információ értelmezett adat, új feltárt ismeret, értesülés, hír, üzenet, tájékoztatás.
Tulajdonságai: értéke, mennyisége, minősége, felhasználhatósága, hozzáférhetősége, érthetősége, védelme.
Az információ mérése: Mértékegysége a bit, hartley, nat.
Az információ kódolása: A kód valamilyen információ kifejezésére, hordozására szolgáló rendszer, a digitális áramkörök vagy számítógép esetében számjegyek.
A numerikus kód karakterkészlete csak számokat tartalmaz.
Az alfanumerikus kód karakterkészlete számjegyeket, betűket és írásjeleket is tartalmaz.
A bináris kódok két szimbólum, a 0 és az 1 által felépíthetők.
Kódrendszerek: BCD kód, Természetes BCD kód, ICAN kód, kilences komplemens, decimális szám kilences komplemense, Stibitz kód, ASCII
A gépi információ ábrázolása: A BCD kódban (Binary Coded Decimals) a 10-es számrendszerben felírt számok jegyeit karakterként ábrázolják.
Számábrázolás módok:
Fixpontos kódok. Itt a jelsorozatban a jegyek helyiértéke (a tizedespont) rögzített.
Lebegőpontos kódok. Itt a számot két komponens, a mantissza és a karakterisztika kódolja.
Kódolt számábrázolás
A számítógép felépítése (Neumann elv):

Alacsony és magas szintű programozási nyelvek (generációk).
1. gépi kód
2. Assembly
3. proceduláris nyelvek (eljárásorientált nyelvek) fortran, pascal, c
3.5. objektumorientált programozás c++ c# JAVA Delphi pyton
4. Specializált nyelvek (problémaorientált)
-Funkcionális: LISP
-Logikai: PROLOG
-Adatbáziskezelt: SQL
5. Mesterséges inteligencia

2. Adatbáziskezelés I.: Hierarchikus, hálós és relációs modellek. Kulcsok a relációs modellben. Kapcsolatok: egy-egy típusú, egy-sok típusú, sok-sok típusú. Anomáliák. Funkcionális függőségek,

tranzitivitás. Normálformák.
Adatbázis fogalma: Az adatbázis tágabb értelemben egy olyan adathalmaz, amelynek elemei egy meghatározott tulajdonságuk alapján összetartozónak tekinthetők. Az adatbázis-kezelőknek meg kell oldani ezen adatok rendezését, a köztük lévő kapcsolat nyilvántartását, az adatokhoz való hozzáférés szabályozását, az adatok védelmét, az integritás megőrzését, az adatok módosíthatóságát, lekérdezését, különféle szempontok szerinti kigyűjtését, válogatását és egyéb statisztikai funkciókat is.
Egy meghatározott témakörrel kapcsolatos információk lehetnek például a vevői megrendelések, számlázási vagy készlet-nyilvántartási adatok stb.

Az adatbázis szerkezete

Az adatbázis az adatok és a köztük lévő összefüggések rendszere, amelyet egymás mellett tárolunk. Nagyon fontos, hogy az adatbázisunk szerkezetét jól megtervezzük, mert a későbbiekben csak így tudunk hatékonyan dolgozni vele.
A tábla a logikailag összetartozó adatokat foglalja össze. A tábla oszlopokból és sorokból áll, amelyeket mezőknek, illetve rekordoknak nevezünk.
A rekord az adatbázis egy sora. Egy rekordban tároljuk az egymással összefüggő adatokat.
A mező az adatbázis egy oszlopa, amelyben az egyedek tulajdonságértékeit tároljuk.
Az elemi adatok a tábla celláiban szereplő értékek, amelyek az egyed konkrét tulajdonságai.
Az egyed az, amit le akarunk írni, amelynek az adatait tároljuk és gyűjtjük az adatbázisban. Az egyedet idegen szóval entitásnak nevezzük. Egyednek tekinthetünk például egy személyt.
Az attribútum (vagyis tulajdonság) az egyed valamely jellemzője. Az egyed az attribútumok összességével jellemezhető. Egy személy egy jellemzője lehet például a neve.
Az egyedre vonatkozóan megadott tulajdonságok összességét egyedtípusnak nevezzük. Egy személy leírható például a nevével, életkorával, testmagasságával, a szeme és haja színével együttesen.
Az egyedre vonatkozóan megadott konkrét tulajdonságokat egyed-előfordulásnak nevezzük. Egy egyed-előfordulás például Kis Ede, aki 29 éves, 183 cm magas, kék szemű, barna hajú.

Kulcsok a relációs modellben

Az adatbázis táblái közötti kapcsolatok az adatok integritásának megőrzés céljából különféle kapcsolattípusokat definiálhatunk.

A táblák közti kapcsolatok az egyedek egymáshoz való viszonyát írják le. Az egyedek közti kapcsolatot háromféleképpen írhatjuk le.

Egy-egy (1:1) kapcsolat: az egyik tábla egy eleméhez a másik tábla pontosan egy eleme kapcsolódik.
Egy-több (1:N) kapcsolat: az egyik tábla egy eleméhez a másik tábla több eleme is tartozhat.
Több-több (N:M) kapcsolat: bármely tábla elemeihez a másik tábla

tetszőleges számú eleme tartozhat.

Nem megfelelően felépített adatbázis esetén az adatszerkezetben különféle anomáliák, ellentmondások keletkezhetnek. Egy relációs adatbázisban a következő anomáliák léphetnek fel:
Bővítési anomália: ha egy rekord felvételekor a már korábban tárolásra került információkat is újra be kell vinni.
Törlési anomália: amikor az elem megszüntetésekor a nem hozzá tartozó információk is elvesznek.
Módosítási anomália: amikor az elemi adat módosulásakor az adatbázisban az elemi adat összes előfordulási helyén el kell végezni a módosítást.

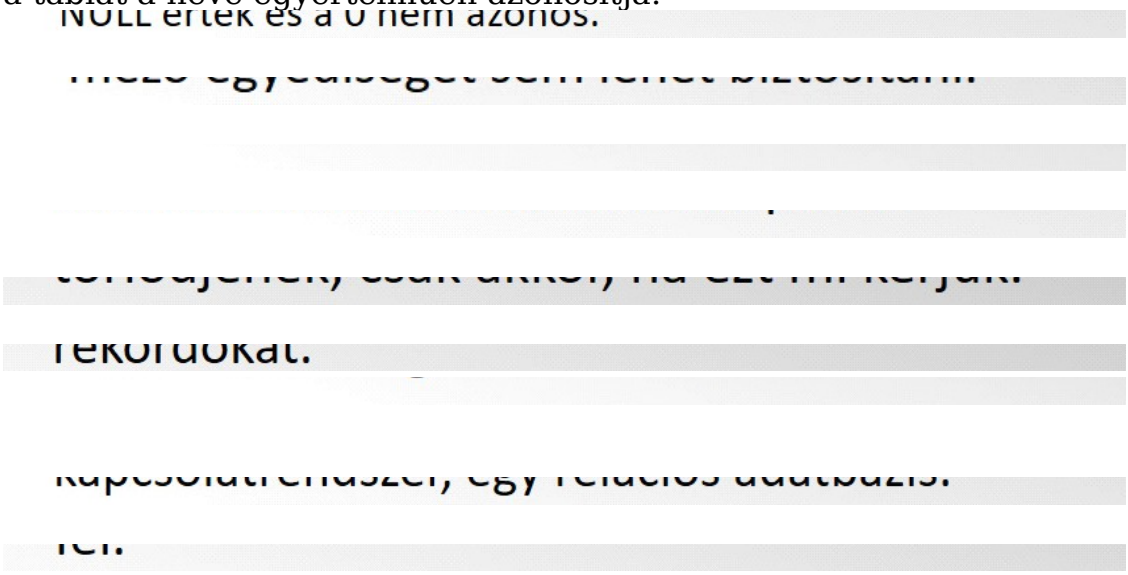
Az adatbázisban előforduló anomáliák, valamint a redundancia (adattöbbszörözés) kiküszöbölésének folyamata a normalizálás. Normalizálással csökkenthető az adatbázisfájl mérete, az adatbázis tartalma logikailag áttekinthetőbbé válik.

A relációs adatmodell

Az adatmodell egyértelműen meghatározza az adatbázis szerkezetét, magában foglalja az adatok típusát, kapcsolatát, a korlátozó feltételeket és az adatkezelési műveleteket.

A mai adatbázisokban négyféle logikai adatmodellt használunk: a hierarchikus, a hálós, az objektumorientált, illetve a relációs adatmodellt. Az Access a relációs adatmodellt használja.

A relációs adatmodellben az adatokat egymással logikai kapcsolatban álló táblákba rendszerezzük. Egy tábla oszlopainak és sorainak a következő feltételeknek kell megfelelniük:
minden oszlopnak egyértelmű neve van,
minden sorban ugyanazok az oszlopok vannak,
az oszlopokban található adatok meghatározott értéket vehetnek fel,
az oszlopok soronként csak egy értéket vehetnek fel,
a táblát a neve egyértelműen azonosítja.



3. Adatbáziskezelés II.: Az SQL nyelv. Relációsémák definiálása. Indexek. Táblák módosítása. A SELECT parancs. Beágyazott lekérdezések. Több táblára vonatkozó lekérdezések. DML parancsok.

Tranzitivitás: ha egy tábla egy mezőjének konkrét értékei meghatároznak a táblán belül más mező értékeit.

Tranzitív függés Tranzitív függés esetén minden, nem kulcs mező függ a kulcstól, de van olyan mező, esetleg mezők, amely a kulcson kívül más mezőtől is függnek. A teljes funkcionális függés egyik feltétele, hogy "minden nem kulcs mező csak a kulcstól függjön, ". Amennyiben ez a feltétel nem teljesül, tranzitív függésről beszélünk. Azt a mezőt, amelytől más mezők tranzitíven függnek, tranzitív kulcsnak hívjuk. A normalizálás során a tranzitív függést meg kell szüntetni!

DML
Az adatkezelési utasítások az egyes rekordok tartalmára vonatkoznak. Lekérdezhetjük egy tábla egy rekordját, több rekordját, összetett rekordokat, amik táblák összekapcsolásával állnak elő. Módosíthatjuk a táblák rekordjait, akár egyszerre többet is, bizonyos feltételeknek megfelelően, és törölhetünk a táblákból.
Fontosabb DML utasítások:
SELECT - adatok lekérdezése
INSERT - adatok felvitele
UPDATE - adatok módosítása
DELETE - adatok törlése
Példa:
Lekérdezés táblák összekapcsolásával, feltétellel, rendezéssel

```
SELECT
  w.Name AS WorkerName, p.Name AS WorkPlaceName
FROM
  workers w
```

```
  LEFT JOIN workplaces p
    ON (w.WorkPlaceID = p.WorkPlaceID)
WHERE
  (w.Born <= 1980) OR (p.Income >= 20000)
ORDER BY
  p.Name, w.Name;
```

SELECT utasítások
Ha egy adatkészletet szeretne leírni az SQL nyelv segítségével, ehhez SELECT utasítást kell írnia. Egy SELECT utasítás az adatbázisból kinyerni kívánt adathalmaz teljes leírását tartalmazza. Ez a következőket foglalja magában:
Mely táblák tartalmazzák az adatokat.
Hogyan kapcsolódnak egymáshoz a különböző forrásokból származó adatok
Milyen mezők vagy számítások eredményeképpen jönnek létre az adatok.
Feltételek, melyek alapján az adatok bekerülnek az eredmények közé.
Szükséges-e az eredményeket rendezni, és ha igen, hogyan.

A SELECT parancs
A SELECT záradék
SELECT [E-mail Address], Company
Ez a SELECT záradék. Egy műveletből áll (SELECT), melyet két azonosító ([E-mail-cím] és Cég) követ.
Ha egy azonosítóban szóközök vagy más speciális karakterek szerepelnek (például az E-mail cím azonosítóban), szögletes zárójelbe kell tenni.
A SELECT záradékban nem kell feltétlenül megadni, hogy a mezők melyik táblában vannak, és nem lehet meghatározni, hogy az adatokat milyen feltételek szerint választja ki.
A SELECT záradék mindig a FROM záradék előtt áll a SELECT utasításban.

Az SQL a legfontosabb nyelv, amely lehetové teszi relációs adatbázisok leírását illetve módosítását is. A legutóbbi SQL-szabvány az SQL-99. A forgalomban lévo adatbázis-kezelo rendszerek túlnyomó része a szabványhoz hasonló - de nem teljesen azonos - dolgokat valósít meg. Az SQL-nek két szempontnak megfelelo része van:
1. az adatdefiníeós résznyelv az adatbázissémák megadásához, illetve
2. az adatmanipuláeiós résznyelv az adatbázis lekérdezéséhez (vagy az adatbázisnak adható kérdések megfogalmazásához), illetve az adatbázis módosításához.
A legtöbb programozási nyelvben megvan egy ilyen típusú megkülönböztetés. Például a C-nek és a Javának is van deklarációs része, illetve futtatható kód része. Ez pedig pontosan megfelel az adatdefiníciós, illetve adatmanipulációs kettosnek. Ebben a részben az adatdefiníciós rész tárgyalását kezdjük el, de részletesebben a 7. fejezetben térünk vissza rá, foként az adatokon teheto megszorítások témakörénél. Az adatmanipulációs résszel a 6. fejezet részletesen foglalkozik.
Az SQL három típusú relációt ismer:
1. Tárolt relációk vagy más néven táblák. Általában ilyen relációkkal foglalkozunk. Ezek benne vannak az adatbázisban, soraik változtatásával megváltoztathatók és soraik is lekérdezhetok.
2. A Nézetek számításokból kapott relációk. Ezeket nem tároljuk, de részben vagy teljes egészben szerkeszthetjük oket, ha ez szükséges. Ezek lesznek a 8.1. alfejezet fo témái.

3. Ideiglenes táblák, amelyeket az SQL nyelvi feldolgozója készít, amikor valamilyen lekérdezéseket és adatmódosításokat végez el. Ezeket a relációkat eldobja és nem tárolja sehol az SQL feldolgozója.
Ebben a részben a táblák megadásának módjait sajátítjuk el. Itt nem foglalkozunk a nézetek meghatározásával és definiálásával sem. Az ideiglenes táblák pedig sohasem deklaráltak. Az SQL CREATETABLEutasításával határozható rneg egy tárolt reláció sémája. Megadja a tábla nevét, az attribútumait, illetve az attribútumok típusát. Emellett kulcs, illetve kulcsok megadását is lehetové teszi. A CREATETABLEutasításnak az eddig említetteken túl több lehetosége is van. Ilyenek például a különféle megszorítások, amelyeket szintén meghatározhatunk vele. Illetve ilyenek az indexek (vagy olyan adatstuktúrák, amelyek a táblán végezheto muveletek felgyorsítására szolgálnak) megadása is.

Indexek létrehozása
A relációkhoz indexeket is rendelhetünk, melyek helyes megválasztása esetén a lekérdezések felgyorsíthatók. Az indexek létrehozására a következő utasítás szolgál:
CREATE [UNIQUE] INDEX index_név ON reláció (attribútum, attribútum, ...);
Egyedi index létrehozása
Az index létrehozásánál a UNIQUE módosító megadásával a reláció valamennyi sorában az index kulcsnak különbözőnek kell lennie. Általában a reláció kulcsok esetén használható csak (index kulcs = reláció kulcs).
Index törlése
Az indexek törlése a DROP INDEX index_név ON [reláció] paranccsal történik. Az indexeket létrehozásuktól a megszüntetésükig az adatbázis-kezelő automatikusan frissíti, a módosításoknak megfelelően. Ha növeljük az indexek számát, a módosítások végrehajtási ideje növekszik.

Táblák módosítása:
Új oszlop hozzáadása: ALTER TABLE táblanév ADD (oszlopnév típus [táblafeltétel]);
Oszlop módosítása: ALTER TABLE táblanév MODIFY (oszlopnév [táblafeltétel]);
Oszlopok törlése: ALTER TABLE táblanév DROP (oszlop, oszlop, ..., oszlop);
Az oszlpok törlését nem minden rendszer engedi meg!
Tábla törlése: DROP TABLE táblanév;

Beágyazott lekérdezések
Az SQL nyelv lehetőséget biztosít arra, hogy a lekérdezések feltételében is használhatunk SQL lekérdező parancsot. Ilyenkor a külső és a belső SELECT parancsot megkülönböztetünk, és az ilyen jellegű lekérdezéseket beágyazott lekérdezéseknek(=alkérdésnek) nevezzük.
Főkérdés – a legkülső szintű kérdés, amely alkérdést (beágyazott kérdést) tartalmazz.
Korrelált kérdés – olyan alkérdés, amely a főkérdés által kiválasztott minden

sorra újra végrehajtódik. Az alkérdés végrehajtásának a kimenete az adott sortól függ.
A SELECT parancsok szintaxisa megegyezik a már ismert formákkal, csupán arra kell ügyelnünk, hogy a belső lekérdezésekre vonatkozóan bizonyos megkötéseknek kell teljesülnie.
A belső lekérdezés jellege alapján következő eseteket különböztethetünk meg:
· A belső lekérdezés egyetlen értéket szolgáltat. Ez a legegyszerűbb eset, ugyanis ilyenkor minden a hagyományos módon történik, azzal a különbséggel, hogy a feltételként megadott kifejezésben a belső lekérdezés által szolgáltatott értéket használja fel a rendszer.
· A belső lekérdezés egyoszlopos relációt szolgáltat. Ekkor olyan feltételeket adhatunk meg, amelyek a belső lekérdezés által visszaadott oszlop adatait használja fel. Ebben az esetben különböző predikátumokat használhatunk.
a). <oszlopkifejezés> [NOT] IN <belső lekérdezés>
Ennél a típusnál az <oszlopkifejezés> paraméterben megadott kifejezés értékéről fogja eldönteni a rendszer, hogy szerepel-e a belső lekérdezés által előállított oszlop adatai között. Ha szerepel, akkor a feltétel értéke igaz, ha nem, akkor hamis lesz. A NOT kulcsszó hatására a feltételt fordítva kell értelmezni.
b). [NOT] <oszlopkifejezés> <reláció művelet> ALL|ANY <belső lekérdezés>
Ennél a típusnál az <oszlopkifejezés> paraméterben megadott kifejezés értékére vonatkozóan azt fogja vizsgálni a rendszer, hogy a megadott reláció művelet teljesül-e a belső lekérdezés által előállított oszlop adataira. Ha az ALL kulcsszót használjuk, a feltétel csak akkor lesz igaz, ha a reláció művelet az oszlop minden elemére teljesül, míg az ANY használatakor elegendő egyetlen elemre teljesülnie. A NOT kulcsszó a feltétel ellentettjét jelenti.

A legáltalánosabb eset az, amikor a belső lekérdezés eredménye általános reláció lesz. Ekkor csak kétféle feltételt alkalmazhatunk.
a). Az egyik az, hogy a keletkezett reláció üres vagy sem. Ehhez a vizsgálathoz az EXISTS kulcsszót kell használnunk, amit természetesen a NOT módosíthat. Az opció formája az alábbi:
[NOT] EXISTS <belső lekérdezés>
b). A másik eset hasonló az egyoszlopos relációról leírtakhoz, azzal a különbséggel, hogy a feltételben több oszlopkifejezést adhatunk meg, amelyek egyezését külön-külön fogja vizsgálni a rendszer a belső lekérdezés által adott tábla sorainak elemeivel. Természetesen a megadott lista és a lekérdezés eredménytáblája sorainak száma azonos kell, hogy legyen. A szintaxisa a következő:
(<oszlopkifejezés> [,<oszlopkifejezés>]…) [NOT] IN <belső lekérdezés>

Több táblára vonatkozó lekérdezések
Legtöbb esetben az adatbázis szerkezete olyan, hogy a szükséges információk több táblában találhatók. Különösen igaz ez, ha normalizált relációkkal dolgozunk, hiszen mint láttuk, a normalizálás alaptevékenysége a több relációra bontás. Ilyen esetekben az adatok összegyűjtéséhez minden táblára szükségünk van, ezért a lekérdezéseket ki kell terjeszteni az úgynevezett többtáblás lekérdezésekké, melyeknek az általános formája a következő:
SELECT [ALL|DISTINCT] * | <oszlopnévlista> FROM <táblanév> [<másodnév>][,<táblanév> [<másodnév>]]…
A kérdés az, hogyan kell értelmeznünk azt, amikor a FROM kulcsszó után több tábla neve szerepel. A választ a relációs modell adja. Ilyen esetben a táblákon a Descartes szorzat relációalgebrai művelet hajtódik végre, s az eredményt a szorzat reláció adja. A gyakorlatban általában egy Descartes szorzat nagyon ritkán használható. Például, ha az egyik táblában a dolgozók adatai, a másikban a fizetési adatok találhatók, akkor ezek Descartes szorzatában minden dolgozóhoz hozzárendelődik az összes kifizetés, az is, ami nem az adott dolgozóhoz tartozik. Ennek nincs semmi értelme, mivel az eredménytábla felesleges sorokat tartalmaz, és azokat el kell távolítani a szorzatból. Erre használhatjuk a kiválasztó műveletet, vagyis a WHERE záradékot, amely tartalmazza a megfelelő feltételt.
Amennyiben a lekérdezéseinkben több tábla szerepel, akkor előfordulhat, hogy valamely attribútumnak mindkét táblában ugyanaz az neve. Ebben az esetben az attribútum neve elé oda kell írnunk a megfelelő tábla nevét.
Ugyancsak előfordulhat, hogy egy táblát saját magával kell összekapcsolni. Ilyenkor a tábla önmagával való Descartes szorzata képződik. Azonban itt az attribútumok nevei mellett magukat a táblák példányait is egymástól el kell különítenünk. Ilyenkor kell megadnunk a másodlagos nevet (másodnevet, alias nevet).
Másodlagos név – olyan ideiglenes név, amelyet egy SQL parancsban egy táblához vagy egy oszlophoz rendelünk hozzá. Ezt a nevet csak az adott parancsban alkalmazhatjuk. A parancson belől a további hivatkozásokban ezt a nevet használhatjuk.

4. Adatszerkezetek és algoritmusok: Algoritmus fogalma, tulajdonságai, megadásának módjai, eszközei, a strukturált algoritmus szerkezete. Programozási tételek: sorozathoz elemi értéket, sorozathoz sorozatot és több sorozathoz egy sorozatot rendelő (kiválogatások, rendező algoritmusok és hatékonyságuk, visszalépéses keresés) algoritmusok. Elemi algoritmusok alkalmazása.

Algoritmuson vagy inkább eljáráson olyan megengedett lépésekből álló módszert, utasítás(sorozato)t, részletes útmutatást, receptet értünk, amely valamely felmerült probléma megoldására alkalmas.

Azt a folyamatot, amikor egy probléma megoldása érdekében meghatározott lépések sorozatát hajtjuk végre, algoritmizálásnak nevezzük

Algoritmus tulajdonságai:

1. Az algoritmus működésének mindig van egy jól meghatározható célja, az algoritmus valójában ezen cél elérésének érdekében dolgozik.
2. A cél elérésének érdekében a feladatokat a számítógép számára egyértelműen definiálható és végrehajtható lépésekre kell bontani.
3. Ebből következik, hogy amíg nem definiálható egyértelműen minden lépés, addig az algoritmus nem működőképes. Például színfelismerés (spektrum-analízis) esetén az algoritmusnak minimálisan egy színhullámhossz-detektorral kell rendelkeznie vagy ilyen készüléktől kell várnia a bemeneti adatokat.
4. Megengedett, hogy nagyobb algoritmikus cél kisebb algoritmikus részfeladatokra legyen bontva.
    a. Például egy számológép-alkalmazás rengeteg további, kisebb algoritmust tartalmaz:
    b. algoritmikus cél: egy számítógépes program, amely egyesíti a legfontosabb matematikai műveleteket, ugyanakkor mindezt felhasználóbarát módon teszi,
    c. kisebb algoritmusok: a konkrét matematikai műveletek, az adatbevitel és megjelenítés, valamint kommunikációs algoritmusok, stb.
5. Az algoritmusnak mindig van egy kiindulási és egy végállapota (más szavakkal belépési és kilépési pontja).
6. A kiindulási és végállapot tulajdonságai különbözhetnek egymástól, ám ez nem kizáró tényező.
7. Az algoritmus mindig véges számú lépésből áll, bár működése tarthat a végtelenségig (például forgalomirányító jelzőlámpa egy fontos közlekedési csomópontban).

Az algoritmus megadásának módjai:

- természetes nyelven (pl. egy ételrecept esetében)
- függvénnyel
    - felsorolással, táblázattal (pl. függvénytáblázatban megadva)
    - grafikusan, függvénydiagram segítségével (pl. koordináta-rendszerben ábrázolva)
    - képlettel, "kiszámítási szabállyal" (visszavezetve elemi műveletekre)
- pszeudokóddal (a természetes nyelv formalizálásával)
- ábrázolva, grafikusan (pl. folyamatábrával)
- programozási nyelv segítségével (pl. a Pascal programozási nyelv eredetileg épp erre a célra készült)
- (absztrakt, matematikai) automatákkal (pl. egy Turing-gép programja

segítségével)

Algoritmus megadásának eszközei

Algoritmusleíró eszközök

1. Folyamatábra
2. Struktogram
3. Jackson-diagramok
4. Leírás fával
5. Leírás programozási nyelven
6. Leírás mondatokkal
7. Leírás mondatszerű elemekkel
8. Leírás absztrakt függvényekkel

Első lépésnél fentről lefelé módszerrel célszerű készíteni, a megoldásnál pedig lentről felfelé. A tervet átültetjük konkrét programozási nyelvekre. A feladat lépéseit a különféle utasítások jelentik:
– I/O utasítás
– értékadó utasítás
– felvételvizsgáló- vagy tesztelő utasítás
– adatátalakító utasítás

A strukturált programozás egy programépítési alapelv. A lényege az, hogy az elvégzendő feladatot kisebb, egymáshoz csak meghatározott módon kapcsolódó részfeladatokra kell felbontani, ezeken belül ugyanígy részfeladatok határozandók meg, amelyek teljes egészében megvalósítják az őket magába foglaló nagyobb részfeladatot, és így tovább. A cél az, hogy a teljes feladat olyan kis feladatelemekre legyen felosztva, amelyek egymással nincsenek átfedésben, egymáshoz meghatározott logika szerint kapcsolódnak, és mindegyik megoldható valamilyen elemi struktúra, elemi programséma követésével, esetleg egy programnyelv saját elemi eszközeivel. A részfeladatokat megoldva és azokat egymáshoz illesztve megszületik a teljes feladat megoldása. A feladat felosztásakor a részfeladatok csak három szerkezeti mintázat szerint kapcsolódhatnak egymáshoz, ezek a szekvencia, a feltételes elágazás (szelekció) és a feltételes ciklus (iteráció vagy véges ismétlés).

A szekvencia alapelem csak annyit jelent, hogy a részfeladatok, illetve azok megvalósítása valamely programnyelvben egymás után következnek, a jelzés szerint felülről lefelé hajtódnak végre. A program elve vagy nyelvi megvalósítása a strukturálás során eljut odáig, hogy a szekvenciában (lépéssorozatban) tovább már nem bontandó elemi műveletek vagy elemi utasítások (statement) követik egymást.

A feltételes elágazás (más néven szelekció) egy megadott feltétel (condition) teljesülését vizsgálja meg (például egy programváltozóban tárolt számérték nagyobb-e száznál), és eszerint folytatódik a program végrehajtása az "igen" (y) és "nem" (n) esetekre előírt műveletekkel

A feltételes ciklus (más néven hurok, ismétlés vagy iteráció) olyan

alapstruktúra, amelyben amíg egy megadott feltétel (condition) teljesül, addig és csak addig végrehajtja a feltételhez rendelt ciklusmag műveleteket, egymást után újra megismételve. Például amikor a könyvespolcon valamilyen sorrendben addig vesszük le, nyitjuk ki, majd tesszük vissza egyenként a könyveket, amíg meg nem találunk benne egy bizonyos részletet.
Az elmélet fontos részét képezi az a szabály is, hogy ezeknek az alapelemeknek a dobozai egymásba helyezhetők tetszőleges mélységig. Ahol tehát a vázlatban egy utasítás (statement) helye látható, oda a három alapelem bármelyikének betehető egy példánya, azokba újabbak és így tovább, szükség szerint

Programozási tételek:

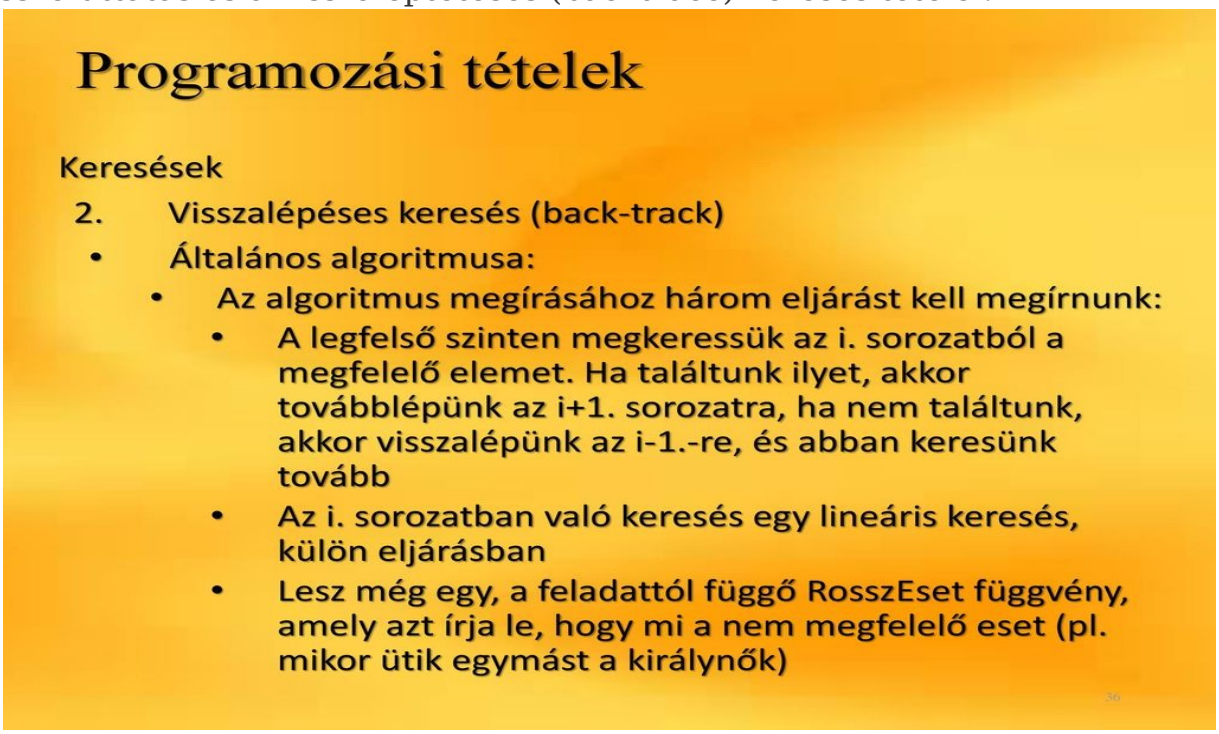
Sorozathoz elemi értéket rendelő algoritmusok

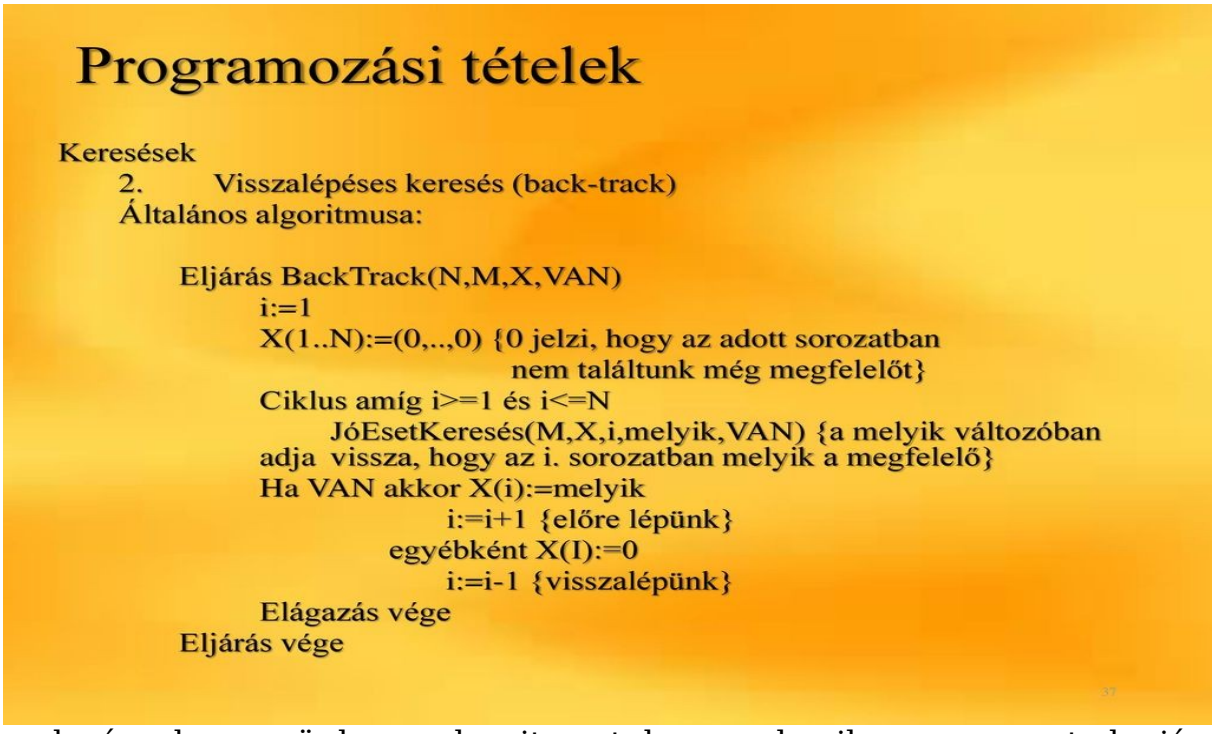
A programozási tétel fogalma: A programozási tételek típusalgoritmusok, amelyek alkalmazásával garantáltan helyes megoldást adhatunk a programozási munka során gyakran előforduló, tipikus programozási feladatokra. A programozási tételek nem egyedi feladatokra, hanem feladatosztályokra adnak megoldást, ezért a megfogalmazásuk is általános. Azokat a típusalgoritmusokat tekintjük programozási tételeknek, melyeknek helyessége bizonyított.
A programozási tételek megfogalmazásának elemei: 1. Feladatkitűzés: az általános feladat szöveges formában 2. Specifikáció: az általános feladatban szereplő adatok (például a függvények) Megadása, elő- és utófeltételek megadása 3. Algoritmus: az általános specifikációban szereplő adatok segítségével megfogalmazott megoldás.
A programozási tételek csoportosítása: A programozási tételek a bemenő és kimenő adatok jellege szerint az alábbi módon csoportosíthatók: - Egy sorozathoz egy értéket rendelő tételek. Ide soroljuk azokat is, amelyek egy sorozathoz néhány egymással összefüggő értéket rendelnek. - Egy sorozathoz egy sorozatot rendelő tételek. (Pl. Rendezések, kiválogatás) - Egy sorozathoz több sorozatot rendelő tételek. (Pl. Szétválogatás) - Több sorozathoz egy sorozatot rendelő tételek. (Pl. Metszet, unió) Eldöntés: Az algoritmus eldönti, hogy van-e a tömbben adott tulajdonságú elem. Amint talál egyet, a ciklus, leáll. Ha a ciklus azért állt le, mert túlléptünk a tömb utolsó, vizsgált elemén is, akkor nem volt benne keresett elem.
Összegzés: Az olyan feladatokat, melyben a sorozat elemeit valamilyen módon gyűjteni kell, összegzéses feladatoknak nevezzük. Kiválasztás: Az algoritmus megadja, hogy a tömbben egy bizonyos elem hol /hányadik helyen/ van. Csak akkor működik, ha biztosan van ilyen elem. Megszámlálás: Megszámolja, hogy a tömbben hány, adott tulajdonságú elem van. Például: negatív számok.
Keresés: Az előzőnél biztonságosabb algoritmus: megadja, hogy van-e olyan elem, és ha igen, hányadik. (többféle kereső algoritmus van)
Szélsőérték-kiválasztás: Minimum vagy maximum érték keresése: A sorozat elemeit tömbbel kezeljük, mert újra szükség lehet ezekre az értékekre - például a legkisebb elem hányadik a sorozatban.

Sorozathoz sorozatot rendelő algoritmusok

Kiválogatás: Ez az algoritmus egy tömb bizonyos tulajdonságú elemeit teszi egy másik tömbbe. DB változó számolja, hogy a másik tömbbe hány elem került.

Rendezés maximum kiválasztással
• Cserélő rendezés (buborék rendezés)
• Beszúrásos rendezés (közvetlen beszúró rendezés)
• Listás rendezés
• Kiválogatás szabálya Keresés rendezett sorozatban:
• Logaritmikus keresés (bináris keresés)
Több sorozathoz egy sorozatot rendelő algoritmus
Rendezetlen sorozatok egyesítése (unio összefésülés)
• Rendezett sorozatok egyesítése (összefutás)
Több sorozathoz egy sorozat hozzárendelése; metszetképzés, egyesítés, az összefuttatás és a visszaléptetéses (backtrace) keresés tételei.





Rendezésnek nevezünk egy algoritmust, ha az valamilyen szempont alapján sorba állítja elemek egy listáját. A rendezési algoritmusoknak két fő típusa van: az összehasonlító (melyek az elemek összehasonlítása alapján állítják elő a rendezett kimenetet) és a nem összehasonlító (melyek az elemek összehasonlítása nélkül képesek erre) rendezések. Egy másik szempont a rendezés stabilitása, azaz hogy az azonosnak ítélt elemek közötti sorrendet megőrzi-e.

A rendezések hatékonyságát, összehasonlítását általában a szükséges összehasonlítások, cserék átlagos és maximális száma és az extra tárigény alapján végezzük.

Összehasonlító rendezések:

Buborékrendezés: A buborékrendezés egy egyszerű algoritmus, amelyet főleg az oktatásban használunk, mivel nem igazán hatékony. Elve, hogy egy „buborékkal” haladva a tömbben több menetben elölről hátra a buborékban szereplő két elemet felcseréljük, ha azok rossz sorrendben vannak.

Gyorsrendezés: Az egyik legnépszerűbb rendezési algoritmus, amely átlagos esetben gyorsabb, mint a többi algoritmus, viszont hátránya hogy a legrosszabb esetben lassú, és nem stabil rendezés.

Rekurzív algoritmus, kettéosztja a rendezendő listát egy kiemelt elemnél kisebb és nagyobb elemekre, majd a részekre külön-külön alkalmazza a gyorsrendezést.

Beszúró rendezés: Az emberi gondolkodáshoz közel álló algoritmus, amely egyszerre egy elemet visz a helyére.

Stabil rendezés: az eredeti sorrend nem változik

Beszúró rendezés bináris kereséssel: Az emberi gondolkodáshoz közel álló algoritmus, amely egyszerre egy elemet visz a bináris kereséssel megállapított helyére. Rendezetten tartott közvetlen elérésű adatbázisokba új elemet így érdemes beszúrni.

A kupacrendezés a használt adatszerkezetről kapta a nevét, a kupacról. Működése során felépíti a kupacot, majd egyesével kiemeli a gyökérelemet, ami a kupac definíciója miatt a legnagyobb/legkisebb elem lesz. Nem stabil rendezés.

Nem összehasonlító rendezések: Ezek az algoritmusok nem használják az elemek között fennálló rendezési műveletet (ha egyáltalán van ilyen) az elemek összehasonlítására, ezért általában valamely más, speciálisabb követelményt támasztanak, ezért általános esetben nem használjuk őket. Jelölések: n az elemek száma, k a kulcsérték mérete. Az algoritmusok feltételezik hogy a kulcsok egyediek és hogy $n << 2^k$ (n lényegesen kisebb, mint 2^k).
Edényrendezés: Az algoritmus során edényekbe, kategóriákba helyezzük az elemeket a kulcsértékük alapján, ott helyben rendezzük őket valamelyik rendezési algoritmussal, majd újra összefűzzük a most már rendezett részlistákat. Közel áll az emberi gondolkodáshoz abban, hogy hatékonyan lebontja a feladatot egyszerűbben megoldhatóakra, és aztán alulról felfelé építi fel az eredményt.
Elemi algoritmusok alkalmazása: Elemi algoritmusnak az igen gyakran, szinte minden programozási feladat megoldása során szükséges algoritmusokat szoktuk nevezni, amelyek egy viszonylag egyszerű, de jól meghatározott feldolgozási vagy kiválasztási feladatot látnak el egy adatsoron vagy adathalmazon. Az elemi algoritmusok általában önmagukban nem képeznek programot, hanem jellemzően más elemi algoritmusok sokaságával, és az azokat összekötő - a megvalósítandó feladattól függő - tapasztó kódrészletekkel kombinálva alkotják a programokat.
Rendezések

- Buborékrendezés (bubblesort)
- Edényrendezés
- Indexvektoros rendezés
- Halomrendezés (heapsort)
- Kiválasztásos rendezés (selection sort)
- Közvetlen beszúrásos rendezés (insertion sort)
- Beszúrásos rendezés bináris kereséssel (algoritmus)
- Leszámláló rendezés
- Gyorsrendezés (quicksort)
- Összefésülő rendezés (merge sort)
- Shell rendezés (Shell sort)
- Számjegyes rendezés
- Topologikus rendezés

Kiválasztások

Maximum és minimum kiválasztás

Keresések

- Lineáris keresés (linear search)
- Logaritmikus keresés (logarithmic search)
- Mélységi keresés
- Szélességi keresés
- Visszalépéses keresés (Backtrack)

- Gráfkereső algoritmus
- Minimax algoritmus

Matematikai algoritmusok

- Euklidészi algoritmus
- Egyenes Illesztés

Hatékonyságvizsgálat táblázatkezelővel

5. Magasszintű programozási nyelvek I: Elemi típusok, konstansok. Összetett és saját típusok. Deklarációk, hatáskörök, élettartam. Operátorok. Típusmódosító operátorok. Automatikus konverziók. Vezérlési szerkezetek.

Típusszerkezet

A típusszerkezete egyszerű. Néhány alaptípus és a tömb áll rendelkezésre. A deklaráció ugyanúgy történik, mint C-ben. Új típusok létrehozására: csak rekord, de később lesz alias/subtype valamit osztály is.

Elemi típusok

Egész típusok:

- i32, int
- u32, uint
- long
- ulong

Lebegőpontos típusok:

- float
- double

Egyéb:

- bool
- char

Literál nélküli:

- handle
- dhandle

A literál nélküli típusokra csak egyenlőség, nem-egyenlőség vizsgálat használható, valamint érték szerinti másolás. Kezdőértékük nullhandle() illetve nulldhandle(), vagyis 0. Ezeket a típusokat a gazda program pointerei helyett használhatjuk. Így a pointereket nem lehet módosítani, még véletlenül se - mert fordítási hibát okoz.

Tömb típus

A tömb típust a seq template típussal hozhatunk létre. Ez csak a fordító számára template! Templatek nincsenek a nyelv jelenlegi verziójában. Az indexelés 0-tól történik, egy u32 típusú kifejezéssel. Az indexhatárokat ellenőrzi, és ha túlindexelés történik, akkor IndexOutOfRangeException kivételt dob.

Rekord típus

Rekord típust a record kulcsszóval lehet bevezetni. Hasonló a C-ben megszokott struct-hoz, csak nem kell pontosvesszőt tenni a csukó kapcsos zárójel után.

Egyetlen megszorítás tartozik a rekordokhoz: egymásba ágyazott rekordok esetén külön declare/export/import blokkba kell kerüljön. Ennek a problémának a megoldása készülőben van.

Típuskonverziók

Típusok közötti átjárás csak konverziós függvényekkel lehetséges. A konverzióks függvények ellenőrzik a típusok értékkészletét, és illegális konverzió esetén ConstraintErrorException kivételt dob.

Változók, konstansok

A változók típusai csak a fentebb említett alaptípusok illetve a tömb lehet. Lehetőség van globális változók használatára is (global blokk). Konstansok nincsenek a nyelvben, alternatíva: függvények.

Kifejezések, operátorok

A kifejezések hasonlóak, mint a C-ben megszokottak. Ami lényeges különbség, az az és és a vagy operátor viselkedése. Az && és a || operátorok mohók (ezeket lehet csak túlterhelni), és a lusta változataik a &&& és a ||| (alternatív jelölések: &&> és a ||>). Lusta és és vagy csak bool típusú kifejezéseket köthet össze.

A számítástechnikában az összetett típusok olyan adattípusok, amelyek felépíthetők a programozási nyelvekben ismert elemi adattípusokból vagy más összetett típusokból. Az eljárás, amivel az összetett típusok készülnek felépítés vagy kompozíció néven ismert. Egy struct a C-ben és a C++-ban jelzi, hogy összetett adatstruktúráról van szó, az adattípust címkézett mezők vagy tagok alkotják. Azért nevezik így, mert a struct kulcsszót kell használni a deklarálásához, ami a structure rövidítése, vagy még pontosabban a felhasználó által meghatározott adatstruktúra rövidítése.

Saját típust a type kulcsszó segítségével hozhatunk létre a főprogram vagy az alprogramok deklarációs részeiben:

type azonosító = típusdefiníció;

Az azonosítónak a korábban már ismeretett szabályoknak megfelelőnek kell lennie. A típusdefiníció lehet többek között:

- egy már létező típus (standard, illetve korábban már létrehozott saját típus)
- lehet felsorolás
- vagy résztartomány

Deklarációk:

Definíció, deklaráció A definíció és deklaráció szinte teljesen azonos fogalmak. Ha változó deklarálásáról beszélünk, akkor azt akarjuk kifejezni, hogy megadjuk a fordító számára egy használni kívánt változó nevét és típusát. Ha változó definiálását emlegetünk változó deklarálása helyett, akkor azzal azt hangsúlyozzuk ki, hogy lefoglalódik egy hely a változónak a memóriában. Egy változót egyszer lehet definiálni, de bizonyos körülmények között előfordul, hogy többször deklaráljuk. Vagyis egyszer történhet egy változó számára helyfoglalás, de lehetséges, hogy többször is tudatnunk kell a fordítóval a változó nevét és típusát. (Például többmodulos programok esetén.)

Hatáskör
A hatáskör a nevekhez kapcsolódó fogalom. Egy név hatásköre alatt értjük a program
szövegének azon részét, ahol az adott név ugyanazt a programozási eszközt hivatkozza, tehát
jelentése, felhasználási módja, jellemzıi azonosak. A hatáskör szinonimája a láthatóság.
Egy programegységben deklarált nevet a programegység lokális nevének nevezzük. Azt a
nevet, amelyet nem a programegységben deklaráltunk, de ott hivatkozunk rá, szabad névnek
hívjuk.
Azt a tevékenységet, mikor egy név hatáskörét megállapítjuk, hatáskörkezelésnek hívjuk.
Kétféle hatáskörkezelést ismerünk, a statikus és a dinamikus hatáskörkezelést.
A futási idő azon részét, amikor egy változó rendelkezik címkomponenssel, a változó élettartamának hívjuk.
Az operátorok olyan szintaktikai elemek, amelyek egyszerű műveleteket írnak elő, vagy kijelöléseket végeznek. A C nyelvben az operátoroknak hét csoportja van, ezek:
1. elválasztó operátorok,
2. értékadó operátorok,
3. aritmetikai operátorok,
4. relációs operátorok,
5. logikai operátorok,
6. bit operátorok,
7. egyéb operátorok

Az operátorok azt határozzák meg, hogy mit kell tenni a változókkal.
Az operátorok olyan (egy vagy több karakterból álló) szimbólumok, amelyek megmondják, hogyan kell feldolgozni az operandusokat.
Típusmódosító operátorok: Az alaptípusokhoz tartozó char, int és double típuselőírásokhoz bizonyos más kulcsszavakat (típusmódosítókat) kapcsolva, újabb típuselőírásokhoz jutunk, amelyek értelmezése eltér a kiindulási előírástóL A típusmódosítók a hatásukat kétféle módon fejtik ki. A short és a long módosítók a tárolási hosszat, míg a signed és az nnsigned az előjel értelmezését szabályozzák.
Az egész típusok lehetnek elő jelesek (sig ned) és elő jel nélküliek (unsigned)
A típusmódosítók nap jainkra szerves részévé váltak a C nyelvnek, hiszen önmagukban típuselőírásként is használhatók. Az alábbiakban (abcsorrendben) összefoglaltuk a lehetséges típuselőírásokat. Az előírások soronként azonos típusokat jelölnek.
char double ennm típusnév float · ·~·ec~m·t m.t, gguaa., ggn long double long int, long, signed long, signed long int sigaaed char short int, short, signed short, signed short int !Jtruct típusnév union típusnév nnsigued char

nnsigued int, nnsigued nnsigued long, unsigued loog int unsigued short, unsigued short int void

Automatikus típuskonverzió Ha egy műveletben két különböző típusú adat vesz részt, akkor a művelet elvégzése előtt megpróbálja azonos típusúvá alakítani őket. A két operandus közül a kisebb kapacitásút fogja átalakítani a másiknak a típusára. Például egy 1 byte-os és egy 2 byte-os egész szám összeadásakor az 1 byte-ost alakítja 2 byte-ossá. (Természetesen nem az eredeti adatot fogja átalakítani, hanem létrehoz egy 2 byte-os területet, ahova beíródik az 1 byte-on tárolt szám.) Szabályok: - az egészek képesek nagyobb kapacitásúvá konvertálódni - az egészek képesek valóssá konvertálódni - a valósak nagyobb pontosságú valóssá - értékadás operátor esetén mindig a jobb oldali operandus konvertálódik a baloldali típusára Egyes nyelvekben, illetve a gyengén típusos nyelvekben egy adat képes kisebb kapacitásúvá konvertálódni, ha az értéke belefér.

Vezérlési szerkezetek: szekvencia, szelekció (if else), iteráció(ismétlés) hátultesztelő ciklus
Először végrehajtja a ciklusmagban lévő utasításokat, majd megvizsgálja a feltételt (bennmaradási feltétel), és ha ez igaz, akkor újra végrehajtja a ciklustörzset. Vegyük észre, hogy ha ez az első lefutásnál már hamis, attól még a ciklusmag mindenképp lefut egyszer.
Elöltesztelő ciklus Először megvizsgálja a feltételt (belépési feltétel), és csak akkor hajtja végre a ciklusmagban lévő utasításokat, ha a feltétel igaz. Vegyük észre, hogy ha kiinduláskor már hamis a belépési feltétel, akkor egyszer sem fut le a ciklustörzs.
Számláló ciklus Az elöltesztelős ciklus egy speciális esete, amikor a megadott ciklusváltozó (általában i-nek nevezik) értéke minden lépés után automatikusan nő eggyel, a kezdőértéktől a végértékig.

6. Magasszintű programozási nyelvek II. Függvények, eljárások. Paraméterátadás.

A függvények típusának megadása típuskifejezésekkel történik, Curry-formában(Ha Curry-formában gondolkozunk, akkor belátható, hogy lényegében minden operátor egyváltozós.
funkcionális paradigma alapfogalmai
Középpontjában a függvények állnak. Más szóval applikatív nyelveknek is hívják ezeket a nyelveket.
Ezekben a nyelvekben a program nem más, mint típusdeklarációk, osztálydeklarációk, függvénydeklarációkés függvénydefiníciók sorozata.
A program része még a kezdeti kifejezés, amelyben tetszőleges hosszúságú, egymásba ágyazott függvény hívássorozat jelenhet meg. Egy ilyen program végrehajtását a
kezdeti kifejezés kiértékelése jelenti. A kezdeti kifejezésben szereplőfüggvények meghívása úgy zajlik, hogy a hívást szövegszerűen a paraméterek figyelembevételével helyettesítjük a definíció törzsével. A

nyelv legfontosabb építőelemei a saját függvények. A függvények törzse kifejezésekből áll. Saját függvényt beépített vagy általunk már korábban definiált függvények segítségével tudunk definiálni. A beépített függvényeket a nyelvi rendszer biztosítja. A függvények alapértelmezetten lehetnek rekurzívak, illetve létrehozhatóak kölcsönösen rekurzív függvények is.
A függvények fő sajátossága, hogy nevük egy értéket képvisel. Típusuk pontosan meghatározott. A függvényekre nevük és paramétereik leírásával hivatkozhatunk. A függvény megadott paramétereken végrehajt egy utasítássorozatot és visszatér egy értékkel. Ezt a visszatérési értéket tekintjük a függvény eredményének és így értékadásban kell szerepelnie.

Az Eljáráselvű programozási nyelvek (másképp "Procedurális nyelvek")alapja a modularitás és a hatókör. Egy eljáráselvű program egy vagy több egységből (unit) vagy modulból (module) áll, minden egyes felhasználó egy kódkönyvtárból dolgozik. Minden modul egy vagy több eljárásból - függvényből, rutinból, szubrutinból, metódusból (az elnevezés az adott programozási nyelvtől függ) áll. Az eljárások annyiban különböznek a függvényektől, hogy nem értéket, hanem valamilyen tevékenységet képviselnek. Ezekre nevükkel és paramétereikkel hivatkozhatunk. A programok során érdemes eljárásokban elhelyezni azokat az algoritmusokat, amiket többször is használunk különböző helyzetekben. Az eljárásokra tagolt program könnyebben áttekinthető, de nagyobb odafigyelést igényel a használatuk. Az eljárásoknál nagy szerepe van a paraméterátadásnak.

Paraméterátadás a C# -ban
A paraméterátadás módja lehet:
- érték szerinti,
- referencia (refmódosító) szerinti,
- eredmény (outmódosító) szerinti.
Van azonban egy negyedik paraméterátadási technika, amelyet tömb paraméterátadásnak hívunk. Így, ha az aktuális paraméterlista egy konvertálható típusú kifejezés, akkor érték szerinti lesz az átadás, egyébként változó paraméterszámú a metódus. Ekkor az aktuális paraméterlista értékeiből létrejön egy tömb-példány, és ebből operál a metódus.
Lusta kiértékelési stratégia: Szöveg szerinti paraméterátadás.
A kifejezésben először a legbaloldalabbi, legkülső redex kerül átírásra, amely azt eredményezi, hogy ha egy kifejezés az függvényhívás, akkor az aktuális paraméterek kiértékelését csak akkor végzi el, ha szükség van rájuk. Mindig eljut a normálformáig, ha az létezik.
Mohó kiértékelési stratégia: Név szerinti paraméterátadás.
A kifejezésben először a legbaloldali, legbelső redex kerül átírásra. Az aktuális paraméterek kiértékelése történik meg először. Gyakran ez a hatékonyabb, de nem biztos, hogy véget ér, még ha a kifejezésnek létezik is

a normálformája.

7. A rendszerfejlesztés technológiája: Az informatikai rendszerek létrehozásának általános lépései. A tervezés során figyelembe veendő szempontok. Az informatikai rendszerek életciklusa. Folyamatmodellek: vízesés, evolúciós, agilis, komponens, iterációs. A Scrum részletes bemutatása. Az objektumorientált tervezés alapjai. Az UML nyelv.

Az informatikai rendszerek létrehozásának általános lépései:
Ismerni kell:
Mi az előállítandó eredmény
Milyen részekre bontható fel a munka
Milyen típusú munkaerőt igényel a feladat

A problémamegoldás általános sémája:
-A feladat felismerése és megfogalmazása
-A helyzet elemzése, megoldási változatok átgondolása
-A megoldás megtervezése
-A szükséges eszközök kivitelezése
-Megoldás az elkészített eszközök használatával

A fejlesztés szakaszai:
-Megvalósíthatósági vizsgálat
-Rendszerelemzés
-Rendszertervezés
-Kivitelezés
-Bevezetés a gyakorlati alkalmazásba

Az informatikai rendszerek életciklusa
-Informatikai stratégia
-Megvalósíthatósági vizsgálat (új probléma)
-Rendszerelemzés
-Rendszertervezés
-Kivitelezés
-Bevezetés
-Üzemeltetés, módosítások

Folyamatmodellek:
-Vízesésmodell Dokumentumvezérelt folyamat, Rögzített lépések sorrendje, Egyik fázisból a másikba csak a fázis teljes befejezése és a befejezést igazoló dokumentum elkészülte után lehet átmenni. Megnyújtja a fejlesztés időtartamát, garancia is a minőségi munkára. Nagy méretű és hosszú élettartamú rendszerek fejlesztésére használják. Akkor érdemes alkalmazni, ha a követelmények már előre jól ismertek.
-V modell
-Szoftverprototípus készítése:mintapéldány, használható: evolúciós rendszerfejlesztés eszközeként, felhasználói követelmények specifikálásakor, nagyobb befektetést igénylő döntés támogatására.
Alapötlete: minél hamarabb kifejleszteni egy kezdetleges implementációt.
Sok iterációs lépésen keresztüladdig fejleszteni, amíg megfelelő nem lesz
Fejlesztésének folyamatai:elemzés, tervezés, kivitelezés, megvalósítás,

validálás, tesztelés, iteráció
-Spirál modell: lépései:tervezés, kockázatelemzés, termékfejlesztés, értékelés.
Minden egyes iteráció során meg kell adni:
Célok, korlátok(pénz idő stb) alternatívák
kockázatok,veszélyek, kockázatcsökkentő stratégiák
eredmények, tervek, döntés a folytatásról
Az első iteráció eredménye a megvalósíthatósági tanulmány
A második iteráció eredménye a követelmények teljes körének a meghatározása.
A harmadik iteráció eredménye a terv
A negyedik iteráció eredménye a tesztelés
-Újrafelhasználáson alapuló fejlesztés
Előnyei:kisebb kockázat, fokozott megbízhatóság, gyorsabb fejlesztés és csökkenő költségek, szabványoknak való megfelelés, szakemberek hatékonyabb alkalmazása
Hátrányai: növekvő karbantartási költségek, eszköztámogatás hiánya, komponens könyvtárak karbantartása, komponensek megtalálása és adaptálása.
-Kész alkalmazási programcsomagok használata
Előnyei: A fejlesztés időtartama lerövidülhet, a projekt haszna hamarabb jelentkezik, a fejlesztő projekt kockázata lecsökkent, a szoftver azonnal hozzáférhető, Magasabb színvonalú dokmentáció, tanácsadás, a szoftvergyártó cég szakmai felkészültsége jobb
Hátrányai: Nem teljesen illeszzkedik a vállalat üzleti folyamataihoz, körülményes adaptálhatóság, gyakran kell új hardvert, szoftvert vásárolni, az egyes modulok kidolgozottsága eltérő színvonalú, rejtett költségek az installálásnál
-Végfelhasználói fejlesztés
EUD End user development
táblázatkezelők, adatbáziskezelők, 4GL, adatelemzző szoftverek
kis méretű feladatokra szorítkozik
Problémái: hosszú, a tervezettet meghaladó fejlesztési idő, magas fejlesztési költségek, kevés kimutatható eredmény. A kész rendszerek nehezen módosíthatók és kielégítetlen felhasználói követelmények. Gyakori a redundáns adattárolás. Új eszközökön régi rendszer valósul meg.
-Agilis szoftverfejlesztés: több feljlesztési metódus összessége (2001 Agile Manifesto)
Elvei: inkább egének és interakció, mint folyamatok és eszközök. Inkább múködő szoftver, mint minden részletre kiterjedő dokumentáció.
Inkább együttműködés a megrendelővel, mint tárgyalások vele. Inkább a változások követése, mint a tervé.
Az agilis módszertan részei: Adaptív szoftver fejlesztés (ASD)
RAD (inkább fejlesszünk, mint tervezzünk
Prototípusok készítése
Dynamic Systems Develop Method DSDM (Az időzítés és a költségvetés szűk
Az ASD kiváltja a tervezést, a becslő együttműködő, tanuló ciklus iterációival iterative programming.
Adaptive, mert könnyen adaptálhatók a változások. Minél könnyebben adaptálunk, annál agilisebbek leszünk.
Crystal (Crystal Clear előre meghatározott áru, kis megvalósítási team-ek projektfelépítési és végrehajtási módszere lehet.
eXtreme Programming

Agilis módszertan részei:
Lean Software development: Nem csinálunk olyat, ami a projekthez nem ad hozzá értéket.
Későn döntünk hamar szállítunk. Csak a megfelelő mennyiségű információ birtokában döntünk és agilisak lévén az ügyfél dönt.
A SCRUM részletes bemutatása
A SCRUM a szoftverfejlesztés egy inkrementális iteratív módszere, amit gyakran használnak az agilis szoftverfejlesztés eszközeként.
Jellemzői:egy keretrendszer, amely magában foglal bizonyos tevékenységeket és meghatározott szerepeket.
Főbb szerepkörei: Scrum master, Product owner, csapat (7fő) felhasználók
Minden futam (sprint) során 2 és 4 hét a csapat egy működő szoftveregységet hoz létre. A futam során megvalósítandó funkciók a Product backlog, bool termék teendőlistája kerülnek ki.
A futamtervező megbeszélés során dől el, hogy a futam során a lista melyik elemeit valósítják meg.
A csapat eldönti, hogy a következő futam során mit tud megvalósítani és azt megígéri.
A futam során a teendőlistát nem lehet megváltoztatni.
A futam végén a csapat bemutatja az elkészült funkciókat.
Alapelve: a megrendelő a fejlesztés során meggondolhatja magát.
A Scrum elfogadja, hogy nincs lehetőség a probléma teljes megértésére és definiálására.
Maximálisan segíti, hogy a csapat gyorsan meg tudja valósítani a funkciókat és gyosan tudjon reagálni.
Megbeszélések:
-Napi Scrum mgbeszélés
-Futamtervező megbeszélés (sprint planning) (a végén futamáttekintés és visszatekintés)
-Az objektum orientált tervezés alapjai az UML nyelv
-Szoftverrendszerek fejlesztésére szolgáló általános célú vizuális modellező nyelv.
-Nem programozási nyelv, nem folyamatmodell, nem módszertan.
-Korábbi technikák szintéziseként alakult ki.
-Szorosan kapcsolódik az USDP-hez
-Szerzők
Diagramtípusok:
Struktúradiagrammok:
-Osztály
-Objektum
-Komponens
-Csomag
-Összetétel
-Telepítés
Viselkedési diagrammok:
-Tevékenység
-Használati eset
-Állapotgép
-Kölcsönhatás szekvencia, kommunikáció és kölcsönhatás áttekintés

UML nyelv Minta egy olyan általános probléma megoldása, amely működött a múltban és hasonló módon újra felhasználható a jövőben.
- Idióma

: a nyelvi mintát hívjuk így, például hogyan kell megnyitni egy állományt.
- Csomag
: típusokat és csomagokat tartalmaz, és egy névteret jelöl ki.
- Osztálydiagram
: objektumokra szaggatja szét a valós világot.